XcalableMPによる HPC Challenge Benchmarksの 実装と性能評価

中尾 昌広

Center for Computational Sciences,
University of Tsukuba, Japan

HPC Challenge(HPCC) Benchmarks

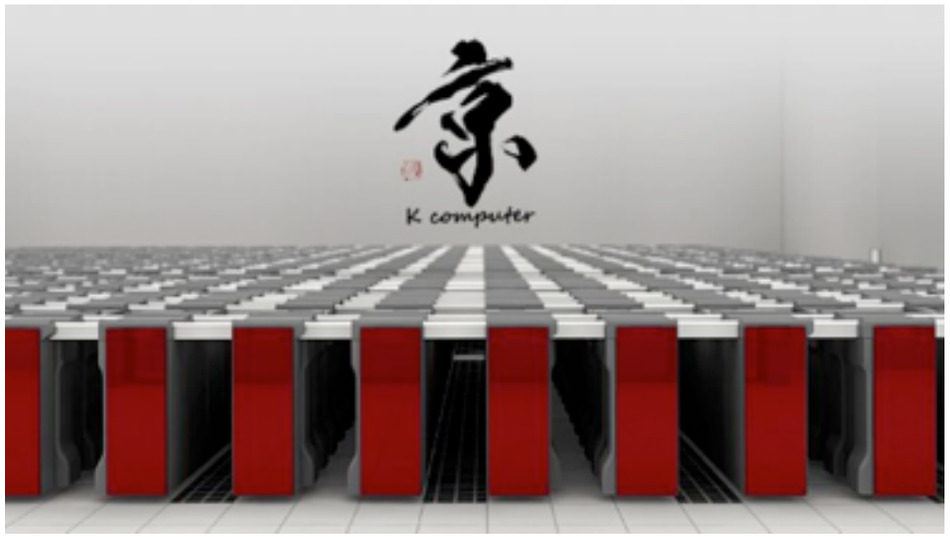
- HPCシステムを多角的に評価するためのベンチマークセット
  - リファレンス実装はオープンソース．規定を満たせば変更可能
- HPCC Award Competition at Supercomputer Conference でも用いられている（毎年10/24が締め切り）

- In Class 1, only the performance of an HPC system is evaluated
- In Class 2, the productivity and performance of a programming language are evaluated
  - STREAM
  - RandomAccess
  - High Performance Linpack (HPL)
  - Fast Fourier Transform (FFT)

- 4つのHPCC Benchmarksの内3つ以上を選択
- 2つまでの，他のベンチマークを追加可能
  - 姫野ベンチマークを追加

Environment

	The K computer
CPU	SPARC64 VIIIfx 2.0GHz 8Cores, **128GFlops**
Memory	DDR3 SDRAM **16GB** **64GB/s/Socket**
Network	Torus fusion six-dimensional mesh/torus network, **5GB/s**



http://www.aics.riken.jp

- 今回の実装はSC13後に公開します（omni xmp compilerのHPで）
- 実装の詳細はPGAS2013で発表（FFTだけ少し異なる．STREAMはない）
  - http://www.pgas2013.org.uk
  - 上記URLで原稿も公開されてます

STREAM：実装

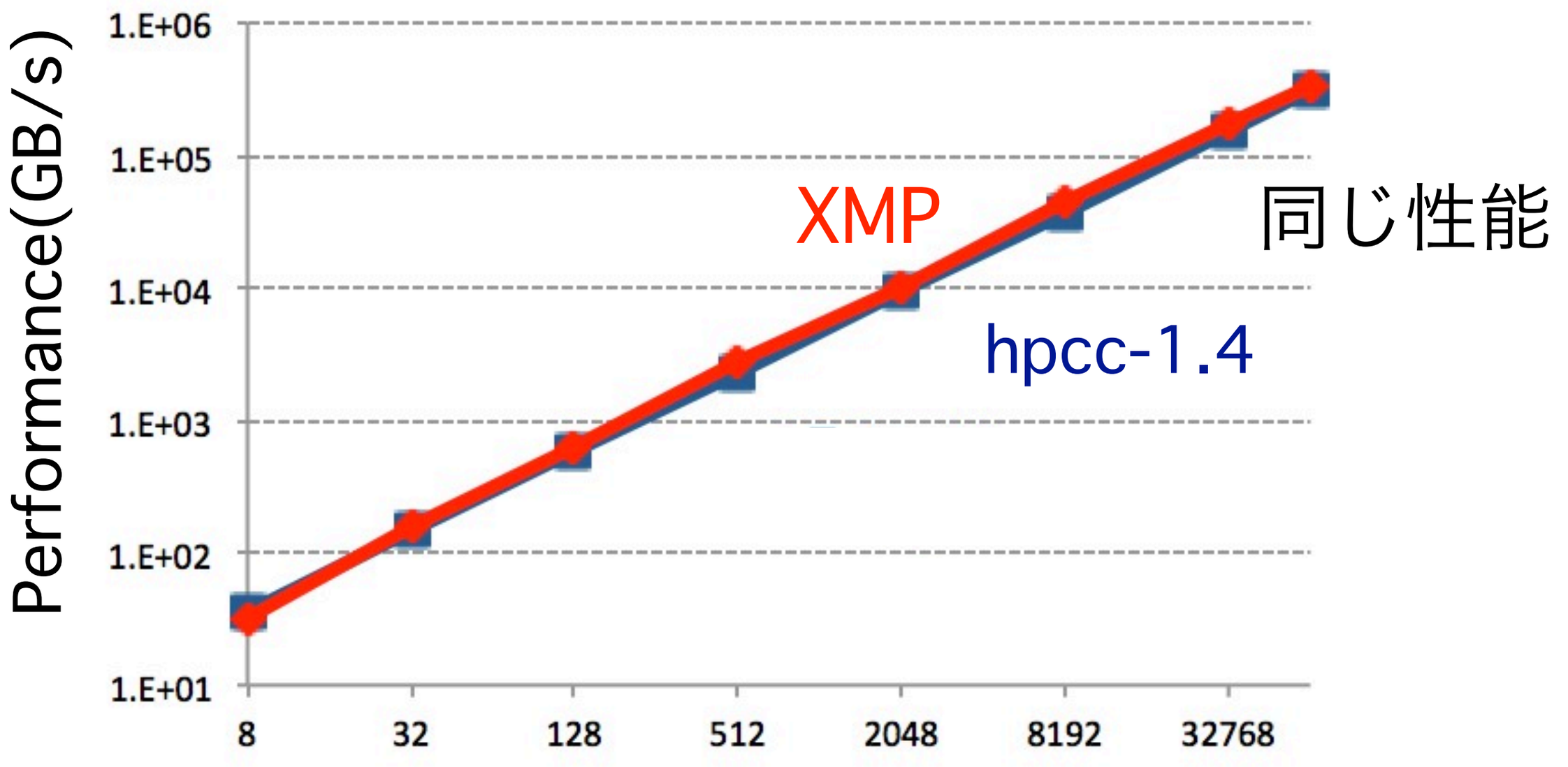
- メモリバンド幅を主に測定
- 非常に単純なベンチマーク．　カーネル部分に通信は発生しない
- 青字がカーネル部分

```
for(k=0; k<NTIMES; k++) {
  times[k] = -xmp_wtime();

  for (j=0; j<size; j++)
    a[j] = b[j] + scalar*c[j];

  times[k] += xmp_wtime();
}
triadGBs =（上の時間からスループットを計算）
#pragma xmp reduction(+:triadGBs)
```

STREAM：性能

flat-MPIで実行

332TB/s for 8192ノード(65536 Cores)



同じ性能

RandomAccess

- The RandomAccess benchmark measures the performance of random integer updates of memory via interconnect
  - 単体ノードの場合は，メモリのランダムアクセスの速度
  - 複数ノードの場合は，ネットワークの速度が重要になる
    - 各プロセスは，他のプロセスのテーブルを更新する
    - ローカルビュー（Coarray）が向いている：Put通信
- 通信回数を減らすため、チャンク毎に通信を行う
  - 相手に何要素送ったかを伝えるために、送信データの最初の要素に要素数を代入
  - post/wait指示文を使って、相手に届いたことを保証

RandomAccess：実装

```
u64Int recv[MAXLOGPROCS][RCHUNK+1]:[*];

for(...){
  ...
  send[isend][0] = nsend;  // set "number of transfer elements"
  recv[j][0:nsend+1]:[send_target] = send[isend][0:nsend+1];
#pragma xmp sync_memory
#pragma xmp post(p(send_target), 0)
  ...
#pragma xmp wait(p(recv_target))
#pragma xmp sync_memory
  nrecv = recv[j-1][0];
  sort_data(&recv[j-1][1], nrecv, ..);
  ...
}
```

Coarrayの宣言

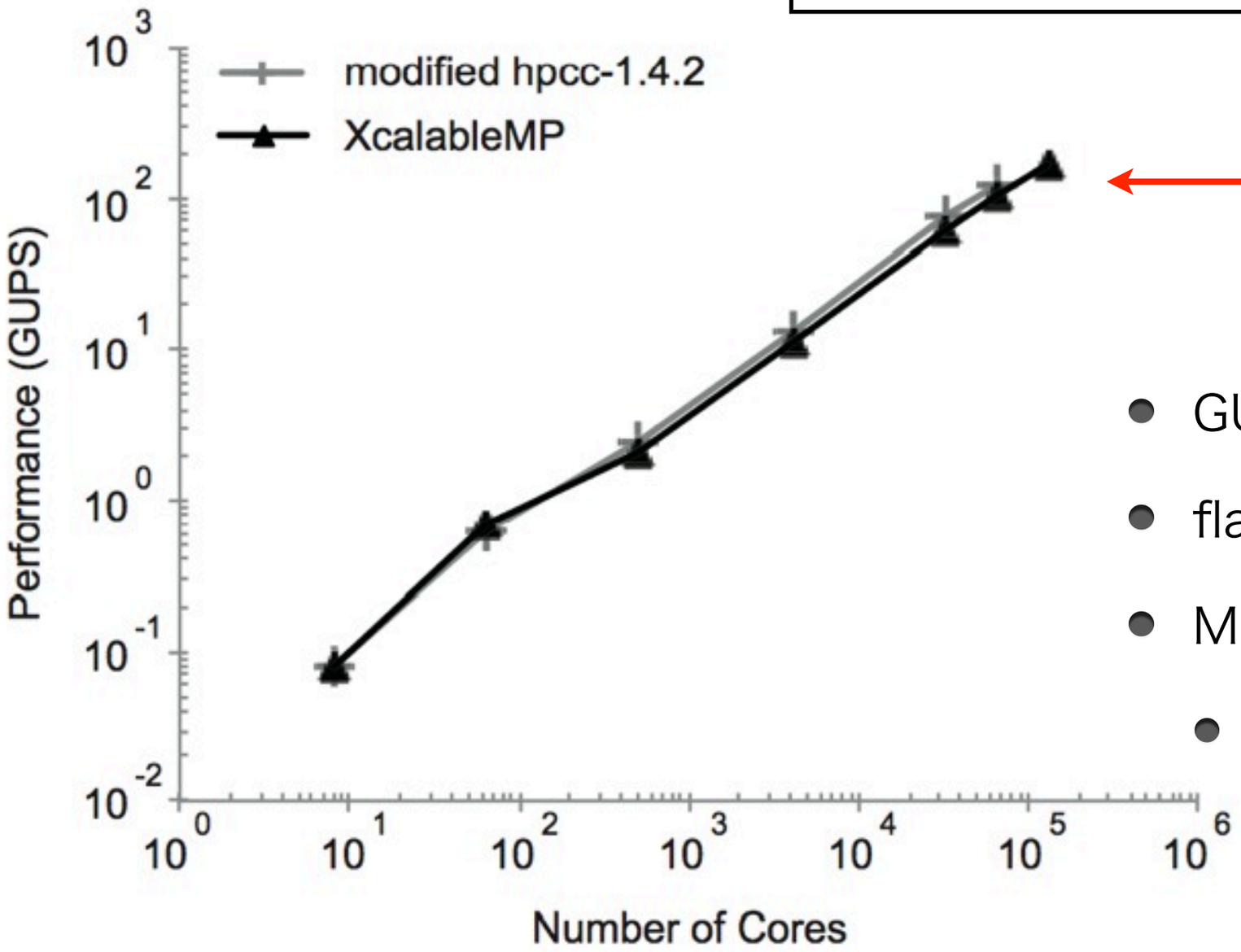
要素数を代入

PUT

PUTの完了を保証

Performance of RandomAccess

The modified hpcc-1.4 RandomAccess,
ローカルの処理を京用に最適化したRandomAccess



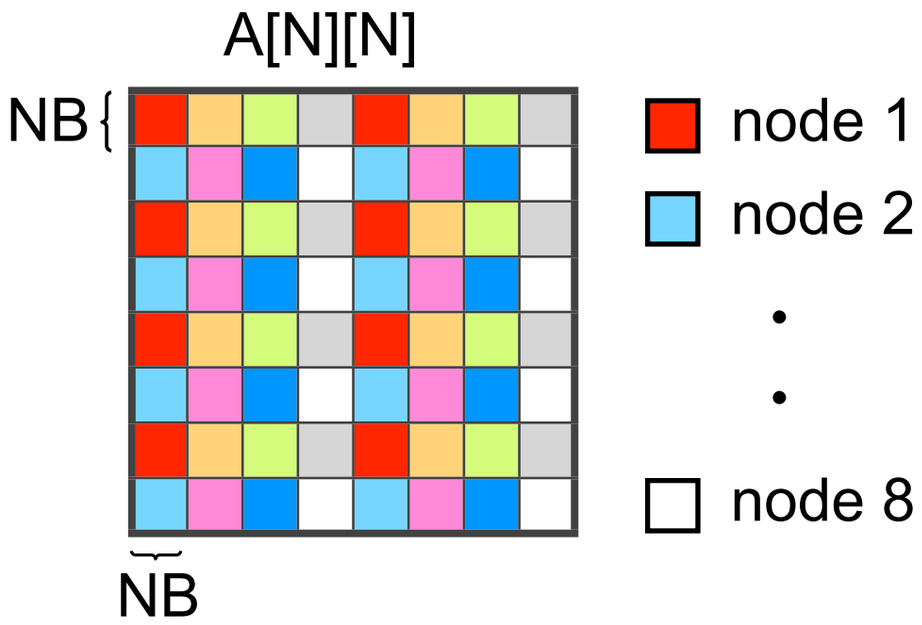
163GUPS in 16,384 nodes
(131,072 CPU Cores)

- GUPS (Giga UPdates per Second)
- flat-MPI (8 Process/Node)
- MPIの方が15%ほど性能高い
  - post/waitの実装を改善中

High Performance Linpack (HPL)

- 密行列の連立一次方程式をLU分解で解く
- Top500でも用いられている．浮動小数点演算性能が強く影響する
  - 係数行列を2次元ブロックサイクリック分割する（hpcc-1.4と同じ）
  - DGEMMには京が提供しているBLASライブラリを用いる

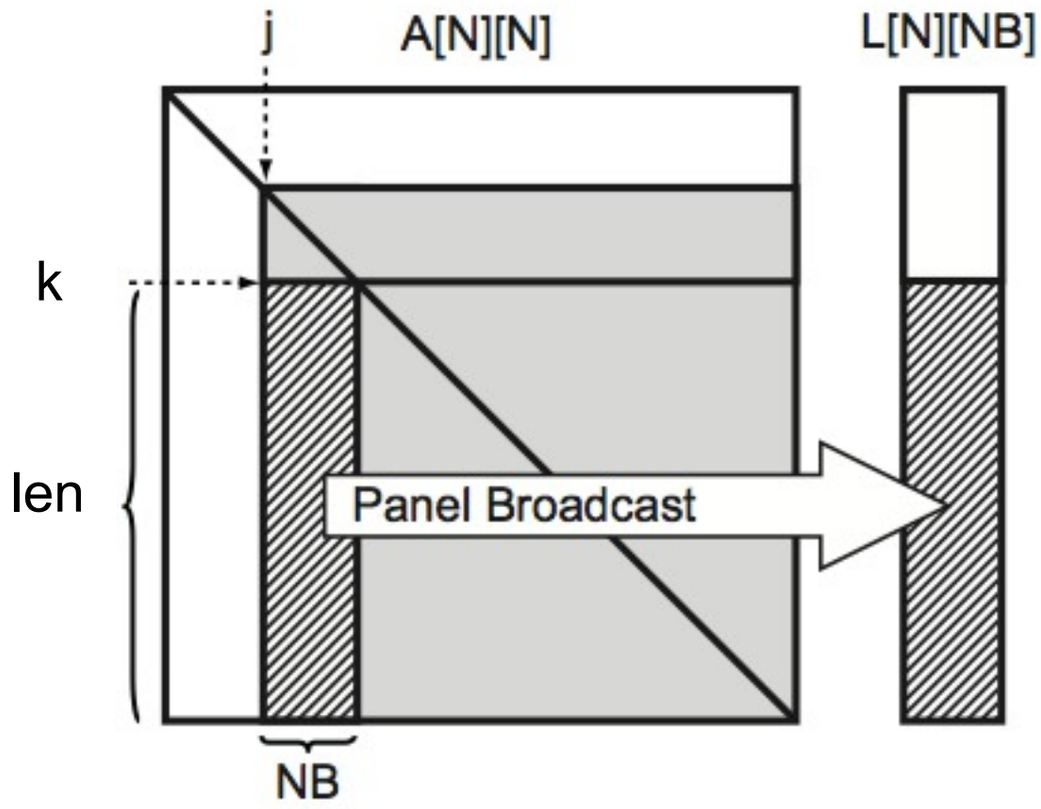
```
double A[N][N];
#pragma xmp nodes p(4,2)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), \
                         cyclic(NB)) onto p
#pragma xmp align A[i][j] with t(j,i)
```

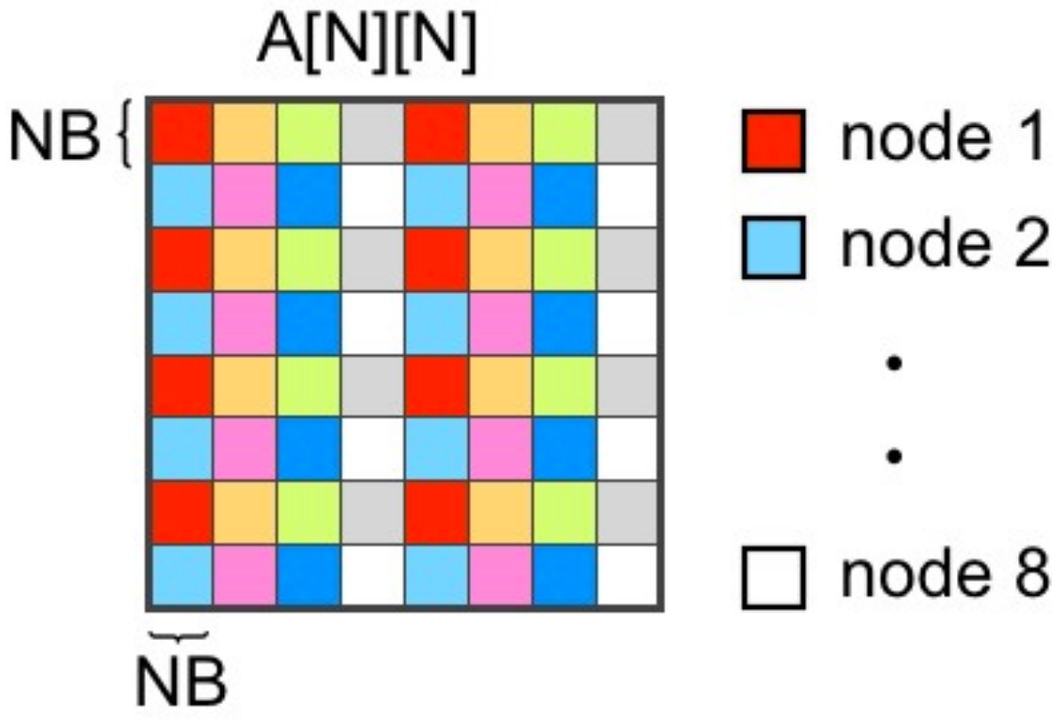


High Performance Linpack (HPL)

- **gmove**指示文を用いたパネル転送

```
double A_L[N][NB];
#pragma xmp align L[i][*] with t(*,i)
     :
#pragma xmp gmove
L[k:len][0:NB] = A[k:len][j:NB];
```



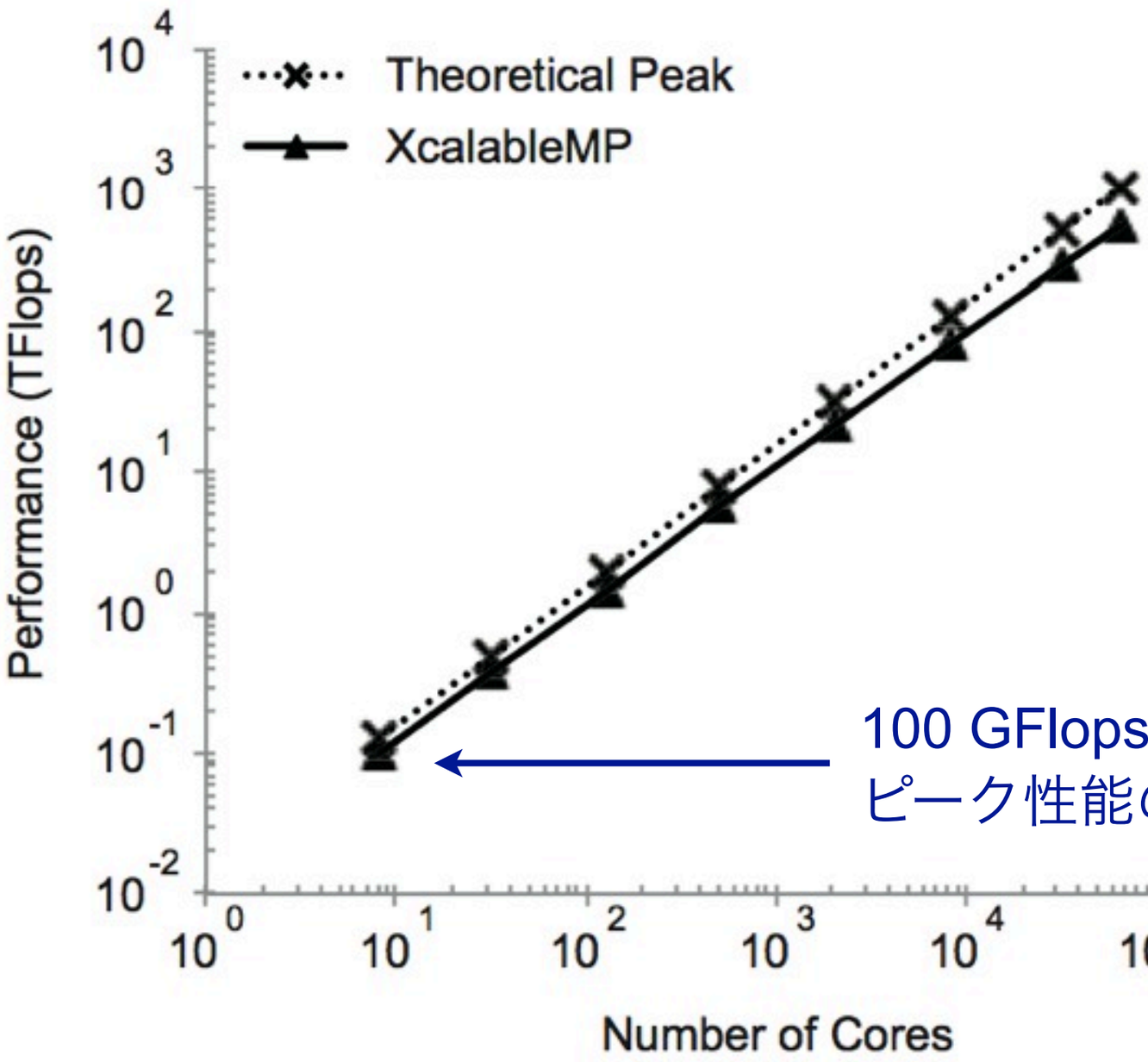


■ は ■ □ ■ にデータを転送する

■ は ■ □ ■ にデータを転送する

Performance of HPL

- 8 Threads/Process on 1 node



543 TFlops in 8,192 nodes
(65,536 CPU Cores),
ピーク性能の53%

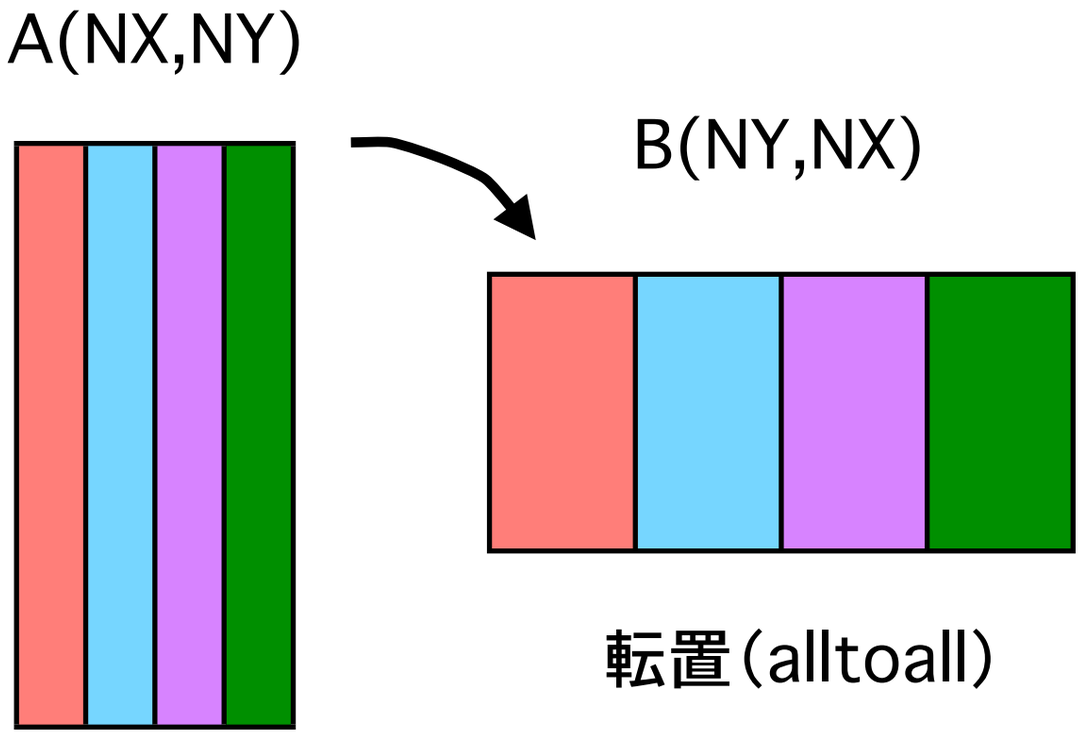
100 GFlops in 1 node, 8 CPU Cores,
ピーク性能の78%

Fast Fourier Transform (FFT)

- FFT measures the floating point rate of execution for double-precision complex one-dimensional Discrete Fourier Transform
- FFTEライブラリ（by 高橋先生@筑波大）
  - http://www.ffte.jp
  - 京用に最適化（ Daisuke Takahashi, Atsuya Uno, and Mitsuo Yokokawa. “An Implementation of Parallel 1-D FFT on the K computer”, IEEE 14th International Conference on High Performance Computing and Communications, 2012）
  - MPIで記述されたFFTEのmain kernelの1つをXMP化
- PGAS2013との違い
  - 行列の転置に組込関数 XMP_TRANSPOSE()を利用
    - 内部で行われるパッキングなどはスレッド並列化

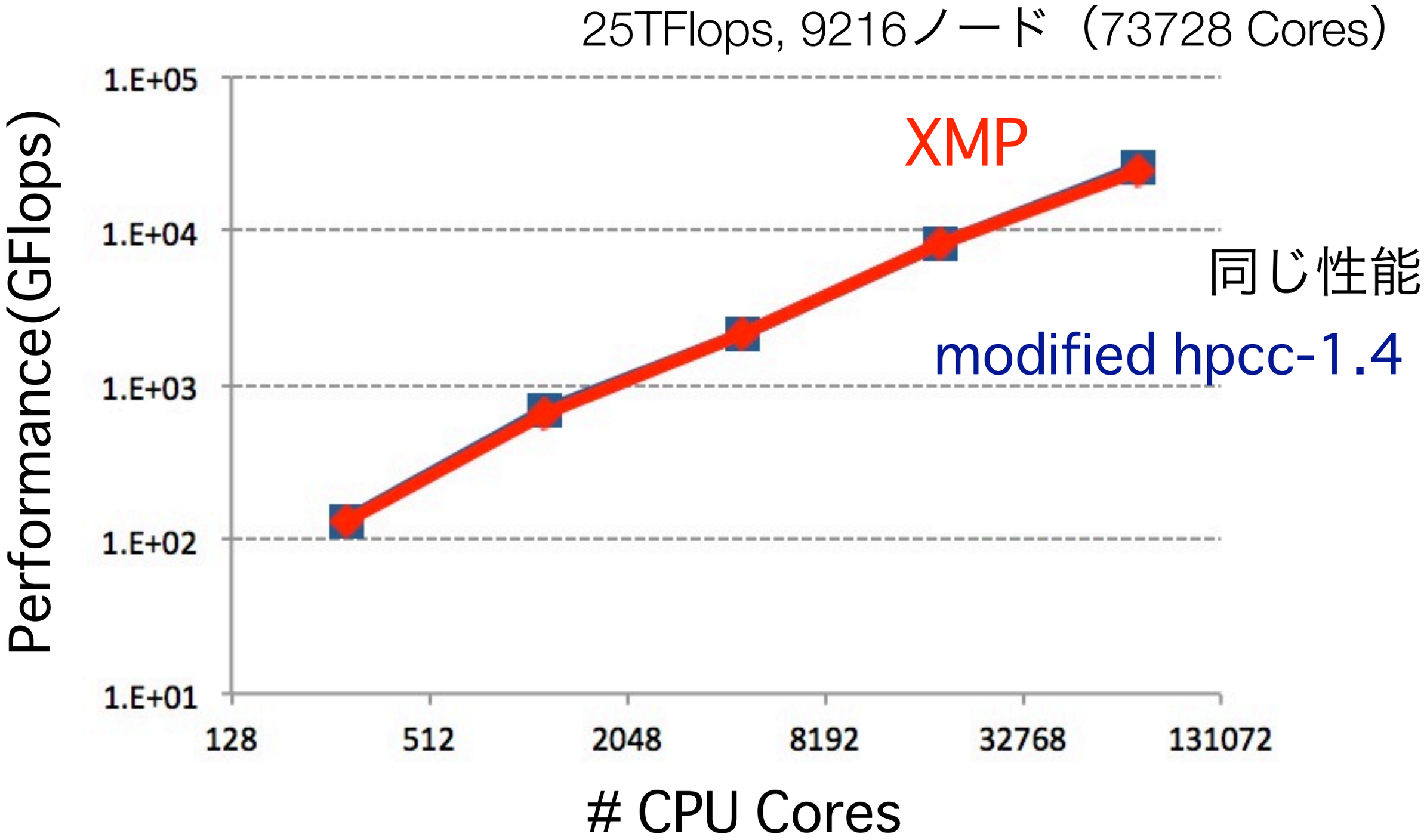
Fast Fourier Transform (FFT)

```
!$XMP template tx(NX)
!$XMP template ty(NY)
!$XMP distribute tx(block) onto p
!$XMP distribute ty(block) onto p
!$XMP align A(*,i) with ty(i)
!$XMP align B(*,i) with tx(i)
!$XMP align W(*,i) with tx(i)
  ...
CALL XMP_TRANSPOSE(B,A,1)
  ...
!$XMP loop on tx(I)
!$OMP parallel do
DO I=1,NX
   DO J=1,NY
      B(J,I)=B(J,I)*W(J,I)
   END DO
END DO
```



OpenMP + XMP

Performance of FFT



姫野ベンチマーク

- 非圧縮流体解析コードの性能評価のためにポアッソン方程式解法をヤコビの反復法で解く場合に主要なループの処理速度を計るもの
- ステンシル計算
- reflect/shadow指示文（京用の高速化）

姫野ベンチマーク

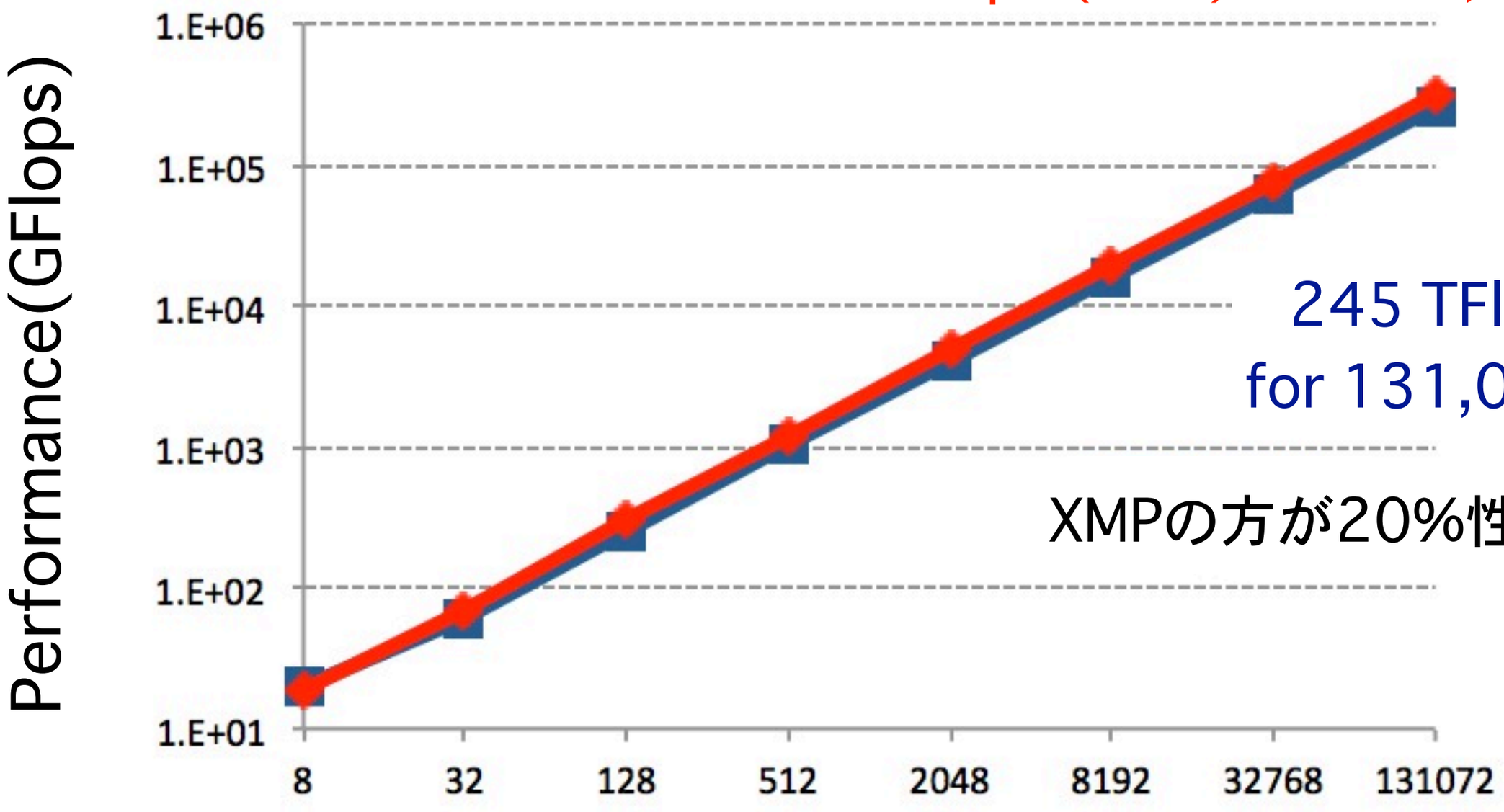
```
!$xmp shadow p(0,2:1,2:1)                                  袖の定義
...
!$xmp reflect (p)                                          袖の同期
DO loop=1,nn
  ...
!$xmp loop (J,K) on t(*,J,K)
  DO K = 2, kmax-1
    DO J = 2, jmax-1
      DO I = 2, imax-1
        S0 = a(I,J,K,1)*p(I+1,J,K) + ...
        SS = (S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
        GOSA = GOSA + SS * SS
        ...
      enddo
    enddo
  enddo
  ...
!$xmp reflect (p)                                          袖の同期
!$xmp reduction (+:GOSA)
end do
```

※pは分散配列

逐次の
姫野ベンチマークに
XMP指示文を
追加するのみで
並列化可能

姫野ベンチマーク

299 TFlops (XMP) for 131,072 Cores



245 TFlops (MPI) for 131,072 Cores

XMPの方が20%性能が高い

他のPGAS言語との行数の比較

- Comparison with Other PGAS Implementations & MPI (hpcc-1.4)
  - HPCC Awards class2 in 2011 and 2012

Lang.	STREAM	RandomAccess	HPL	FFT
XMP	66	250	306	（modified hpcc-1.4とほぼ同じ）
Coarray Fortran	63	409	786	450
Chapel	72	112	658	(NO DATA)
X10	60	143	708	236 + FFTE
MPI (hpcc-1.4)	329	938	8,800	1904

まとめ

- STREAM，FFT，RandomAccessはhpcc-1.4ベースとほぼ同等
- HPLの並列化効率はあまり良くない
  - gmove指示文（実装の最適化が不十分）
  - 改善案
    - Coarrayを用いて、ロードバランスが良くなるようにパネル転送のスケジュールを決定する
      - 隣接ノードに先にデータを渡す
    - 実装済み．SC13で発表予定